UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE CIENCIAS
DEPARTAMENTO DE MATEMÁTICA Y CIENCIA DE LA COMPUTACIÓN
LICENCIATURA EN CIENCIA DE LA COMPUTACIÓN
Profesor(es): Nicolas Thériault
Primer Semestre de 2016

**Matemática Computacional – Laboratorio 3**



# 1. Representación de grafos en un computador

La representación grafica de un grafo (dibujo) es práctica para trabajar manualmente, pero no conviene en un ambiente computacional. Existen varias maneras de representar grafos en un computador, las principales son:

1. **Listado de aristas**: Se hace un listado con cada arista $(a, b)$ en el grafo. Para grafos simples (con a lo más una arista entre cada par de vértices) este formato es generalmente lo más compacto, pero su desventaja principal es que no permite un trabajo eficiente ya que al principio no hay orden en las aristas. Algunos de los problemas encontrados son:
   - En grafos no-dirigidos (sin flechas), ¿Cómo escribir (o cómo está escrtia) la arista: $(a, b)$ o $(b, a)$?
   - ¿Cómo asegurar que ninguna arista está repetida?
   - ¿Cómo determinar si hay aristas saliendo de un vértice (sin verificar toda la lista)?
   - ¿Cómo encontrar todas las aristas de un vértice?
2. **Matriz de adyacencia**: Para representar el grafo, se utiliza una matriz $n \times n$ donde $n$ es el número de vértices en el grafo. La entrada de la posición $(a, b)$ corresponde al número de aristas que van desde el vértice $a$ hasta el vértice $b$ (0 si no hay).
3. **Listado de adyacencia**: En realidad, se trata de un listado-de-listados. Para cada vértice, se hace un sub-listado de los vértices conectados a este primero por una arista, es decir un sub-listado de todas los "destinos" de aristas saliendo del vértice. Se puede empezar el sub-listado por un indice que indica cuantas entradas contiene o indicar por "Null" el final del sub-listado.
4. **Listado dinámico (linked list)**: En algunas situaciones y para algunas computaciones, el grafo no queda constante en el tiempo: algunas aristas se deben eliminar o agregar. La técnica más páctica para adaptar el listado de adyacencia es con un listado dinámico, utilizando punteos. En realidad consiste en tres listados: el listado *principal*, el listado de *indices* y el listado (o stack) de *entradas no utilizadas*.
   - El listado de *indices* es un listado del número de fila de la primera arista saliendo del vértice ("Null" si no hay).
   - El listado de *entradas no utilizadas* contiene las filas del listado principal que no sirven en este instante, y se utiliza para encontrar una fila donde escribir una nueva arista.

- El listado *principal* es una secuencia (numerada) de filas que consisten en dos entradas: el vértice final de la arista, y el número de fila de la próxima arista saliendo del vértice de origen ("Null" si la arista actual es la última).

Escribiremos $(d_i, f_i)$ para la $i$-esima fila del listado *principal*, y $F_a$ para la entrada del vértice $a$ del listado de *indices*. Las operaciones que se aplican son:

- Encontrar una arista saliendo del vértice $a$: ir a la fila $F_a$.
- Encontrar la próxima arista saliendo del vértice $a$ (después de la fila $i$): ir a la fila $f_i$.
- Agregar la arista $(a, b)$: Para el vértice $a$, sacar un número $i$ del listado de *entradas no utilizadas*. Aplicar: $f_i \leftarrow F_a$, $F_a \leftarrow i$, $d_i \leftarrow b$. En grafos no-dirigidos, repetir para el vértice $b$.
- Eliminar la arista $(a, b)$: leer las aristas asociadas al vértice $a$ hasta encontrar $d_i = b$. Si $F_a = i$, aplicar: $F_a \leftarrow f_i$. Si $F_a \neq i$, utilizaremos $j$ para el indice anterior (tal que $f_j = i$) y aplicar: $f_j \leftarrow f_i$. En ambos casos, agregar $i$ al listado de *entradas no utilizadas*. En grafos no-dirigidos, repetir para el vértice $b$.

En caso que el número máximo de aristas no es conocido, hay que gestionar como controlar el tamaño del listado principal (si el espacio de memoria es limitado).

El listado dinámico es el más complicado a programar inicialmente, pero es el más flexible y muchas veces permite trabajar de manera más eficiente a largo plazo.

## 2. Ciclo de Euler

Un teorema de Euler (1736) dice que un grafo no-dirigido admite un ciclo que pasa por todas las aristas del grafo exactamente una vez si y solo si está conectado y todos sus vértices tienen grado par. Hoy día, tales ciclos se conocen como *cilcos de Euler*.

Para generar un ciclo de Euler, se inicia un camino hasta no tener más arista para salir del vértice actual (da un ciclo cerrado). Si quedan aristas en el grafo, elegir uno de los vértices del ciclo en lo cual quedan aristas, y "abrir" el ciclo en este punto, repetiendo el proceso.

## 3. Generación aleatoria de grafos

Para experimentar técnicas de trabajo en grafos (grandes), es práctico comprobarlas con grafos generados aleatoriamente que para grafos dados de manera explicita.

Para este laboratorio, generaremos grafos conectados no-dirigidos (1) simples (2) y de grado par (3), es decir: (1) se puede ir desde cualquier vértice hasta cualquier otro vértices pasando por aristas del grafo; (2) entre cada par de vértices hay a lo más una arista (no dirigida) y no hay aristas que vuelven a su vértice de origen; (3) cada vértice tiene un número par de aristas.

El pseudo-codigo para generar estos grafos se encuentra en el Algoritmo 1.

# 4. Laboratorio

- Desarrolle un programa en lenguaje C para generar grafos aleatorios conectados no-dirigidos simples y de grado par y para encontrar un ciclo de Euler en esos grafos.
- Evaluar el costo computacional de la busqueda de ciclos de Euler según el número de vértices $n$ y la probabilidad de aristas $p$.
- Justificar (demostrar) porque el algoritmo para encontrar un ciclo de Euler termina dando un ciclo de Euler.

# 5. Se solicita

1. Implementar en lenguaje C los algoritmos.
2. Determinar los costos computacionales de los algoritmos para $p \in \{1/10, 3/10, 1/2, 7/10, 9/10\}$ para distintos valores de $n$.
3. Determinar los costos computacionales de los algoritmos para $n = 1000$ para distintos valores de $p$.
4. Se debe entregar:
   - Código fuente de los algoritmos.
   - Archivo "Readme" con las instrucciones (detalladas) de compilación.
   - Informe en LaTeX que contiene:
     - Formulación de los experimentos.
     - Breve descripción de la representación de grafos utilizada y porque se eligió.
     - Información del hardware (procesador) y software (librerias) utilizado.
     - Curvas de desempeño de los resultados.
     - Conclusiones.
5. Normas a cumplir:
   *a*) Número de Integrantes: máximo 2.
   *b*) Plazo de Entrega: 14 días.
   *c*) Forma de envío: se envía directorio comprimido zip, gz, tar, o tgz (no se aceptan otro formatos). a: **nicolas.theriault@usach.cl**
6. El nombre del archivo (y directorio) debe indicar el laboratorio ("Lab1") y a lo menos las iniciales (nombre y apellido) de cada integrante.
   - Se puede utilizar solamente el primer nombre y primer apellido de cada integrante.

- No deber haber acentos (á → a, ñ → n, etc).
- No deber haber espacios en blanco, utilizar "_" para separar palabras e iniciales.
- Ejemplo: Rodrigo German Abarzúa Ortiz y Nicolas Thériault → Lab3_RA_NT.

# 6. Evaluación

La nota del laboratorio se calculará según la ponderación siguiente:

- Informe [**20 %**]:
  El informe está en LaTeX, en lenguaje formal, y contiene todos los detalles exigidos.

- Algoritmos y análisis [**40 %**]:
  Justifica/demuestra que el algoritmo de búsqueda encuentra un ciclo de Euler. Estudio de la eficiencia temporal del algoritmo implementado y estimación de como crece (basado en suficientes puntos de evaluación para justificar las conclusiones).

- Implementación [**40 %**]
  El código fuente no presenta errores o advertencias de compilación. El programa está escrito de forma que puede ser leído y/o re-utilizado fácilmente por otros programadores: la redacción es limpia (con espacios y divisiones claras) y bien documentada, las sub-funciones y las variables tienen nombres naturales (que indican a que sirven) o acompañadas de comentarios aclarando a que sirven.

**Algorithm 1** Generación aleatoria de grafos con un ciclo de Euler

**Require:** Número $n$ de vertices del grafo y propoción $p$ de aristas por vertice.
**Ensure:** Grafo $G = (V, A)$ (vertices y aristas) que admite un ciclo de Euler.

1: $V \leftarrow \{1, \ldots, n\}$
2: $A \leftarrow \emptyset$
3: **for** $a$ desde 1 hasta $n - 1$ **do**
4: Incluir $(a, a + 1)$ en $A$
5: **for** $b$ desde $a + 2$ hasta $n$ **do**
6: **if** $Rand() < p$ **then**
7: Incluir $(a, b)$ en $A$
8: **end if**
9: **end for**
10: **end for**
11: $Impar \leftarrow$ {vertices que aparecen un número impar de veces en $A$}
12: **while** $(Impar \neq \emptyset)$ **do**
13: Elegir $a, b$ elementos de $Impar$, $a < b$.
14: **if** $(a, b) \notin A$ **then**
15: Incluir $(a, b)$ en $A$
16: **else**
17: **if** $b \neq a + 1$ **then**
18: Eliminar $(a, b)$ de $A$
19: **else**
20: Elegir un vertice $c < a - 1$ o $> b + 1$.
21: **if** $(a, c) \in A$ **then**
22: Eliminar $(a, c)$ de $A$
23: **else**
24: Incluir $(a, c)$ en $A$
25: **end if**
26: **if** $(b, c) \in A$ **then**
27: Eliminar $(b, c)$ de $A$
28: **else**
29: Incluir $(b, c)$ en $A$
30: **end if**
31: **end if**
32: **end if**
33: Eliminar $a$ y $b$ de $Impar$
34: **end while**
35: **return** $G = (V, A)$